Universidade de Aveiro
2015
Departamento de Eletrónica, Telecomunicações e Informática

JOÃO ANTÓNIO TEIXEIRA COSTA

APLICAÇÃO E SERVIÇOS WEB PARA ANÁLISE DA TOSSE EM PACIENTES CRÓNICOS

Universidade de Aveiro
2015
Departamento de Eletrónica, Telecomunicações e Informática

JOÃO ANTÓNIO TEIXEIRA COSTA

APLICAÇÃO E SERVIÇOS WEB PARA ANÁLISE DA TOSSE EM PACIENTES CRÓNICOS

dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Sistemas de Informação, realizada sob a orientação científica do Dr. Sérgio Guilherme Aleixo de Matos, Professor do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

Dedico este trabalho à minha família e amigos que me apoiaram em todo o meu percurso académico.

o júri / the jury

presidente / president

Prof. Doutor Augusto Marques Ferreira da Silva
professor auxiliar da Universidade de Aveiro

vogais / examiners committee

Prof. Doutor José Paulo Ferreira Lousado
professor adjunto do Departamento de Informática, Comunicações e Ciências Fundamentais da Escola Superior de Tecnologia e Gestão de Lamego do Instituto Politécnico de Viseu
(Arguente Principal)

Prof. Doutor Sérgio Guilherme Aleixo de Matos
equiparado a investigador auxiliar da Universidade de Aveiro
(Orientador)

palavras-chave Aplicação web, serviços web, tosse crónica, análise tosse

resumo O presente trabalho propõe o desenvolvimento de uma aplicação web e os seus serviços capazes de proceder a análise de tosse crónica. Apresenta todas as considerações que foram tomadas no seu desenvolvimento e a explicação das decisões tomadas relativamente ao contexto actual e possíveis alterações futuras para onde a área abordada possa vir a adoptar.

keywords Web application, web services, chronic cough, cough analysis

abstract This paper proposes the development of a web application and its services able to carry out analysis of chronic cough. It features all the considerations that were made in its development and explanation of decisions taken with regard to the current context and possible future changes to where the area covered is likely to adopt.

Conteúdo

Lista de figuras

Lista de acrónimos

Capítulo 1

Introdução

Doenças respiratórias estão entre as principais causas de morte a nível mundial, estimando-se que na Europa morram cerca de 600 mil pessoas anualmente devido a doenças respiratórias. Relativamente a admissões para observação médica, estas doenças representam cerca de 7% do total, totalizando 6 milhões de admissões hospitalares anualmente apenas para a Europa. Relativamente aos custos inerentes, estes ultrapassam os 380 biliões de euros anuais. Importante também referir que o não tratamento da doença poderá provocar uma redução no tempo de vida do paciente [1].

Durante a monitorização e gestão clínica de pacientes com doenças respiratórias, vários estudos clínicos são normalmente realizados para obter vários indicadores da doença e da sua evolução. Estes indicadores são usados tanto para diagnóstico como para determinar a eficácia das possíveis fases de tratamento.

Algumas das mais comuns doenças respiratórias são: Doença Pulmonar Obstrutiva Crónica (DPOC), bronquite, asma e pneumonia. Em todas elas, a tosse é um indicador da doença [2], sendo muitas vezes o primeiro e o mais frequente. Adicionalmente, a sua frequência e gravidade da ocorrência são bons indicadores da alteração do estado do paciente.

É necessária uma medida fiável, de modo a que a gravidade de tosse em um paciente em particular e a eficácia do tratamento possam ser avaliadas. Dada a importância deste sintoma e à falta de testes robustos e objetivos, alguns sistemas foram desenvolvidos recentemente que, de forma automática e não invasiva, permitem obter dados quantitativos de confiança relativos à sua frequência [3][4].

O Leicester Cough Monitor (LCM) é um sistema para monitorização da frequência da tosse suportado por um algoritmo, baseado em Hidden Markov Models (HMM), capaz de detectar de forma automática eventos de tosse, permitindo que os dados sejam recolhidos em regime ambulatório. Desta forma é possível estudar o paciente sem que este tenha de alterar o seu dia-a-dia, devido à capacidade do algoritmo, para ignorar sons que não representam eventos de tosse [3]. Os pacientes devem ser monitorizados durante um período extenso, de forma a obter indicações clínicas efectivas sobre o sintoma. O LCM possui um interface gráfico de forma a interagir com o sistema e realizar introdução de

dados. Com este sistema é possível estudar e monitorizar sintomas de tosse crónica associada a doenças respiratórias [4].

Uma sessão de monitorização desenrola-se em fases distintas, sendo a primeira fase a aquisição de dados do paciente através de um gravador digital portátil de áudio e um microfone condensador miniatura. Esta gravação é feita durante um período longo, usualmente superior a 24 horas.

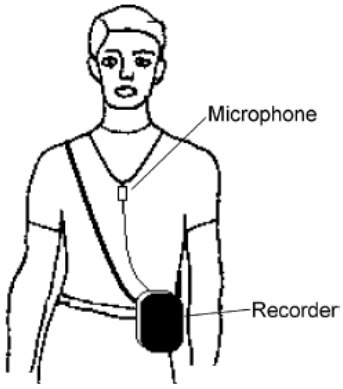

Figura 1 - Sistema de gravação de tosse [4]

A Figura 1 ilustra o sistema utilizado para a aquisição das gravações referentes a um paciente. O áudio é gravado no formato MPEG áudio layer-3 (MP3). Posteriormente à recolha das gravações, estas podem ser submetidas para a fase seguinte que consiste em analisar as gravações. Nesta fase o sistema cria ficheiros wav para cada segmento de áudio que o algoritmo identificou como candidato a evento de tosse. Na fase seguinte usam-se dois modelos HMM, um previamente treinado para reconhecer eventos individuais de tosse e outro para reconhecer outros eventos individuais (não tosse). O que se faz nesta fase é classificar cada candidato como tosse ou não-tosse. Uma vez que os modelos originais são genéricos é utilizado input do utilizador para os adaptar às características de uma dada gravação e obter uma classificação final mais fiável. Essencialmente esta fase faz filtragem (remoção) de falsos positivos identificados na primeira fase, que é menos restritiva. Uma diferença a ter em conta: na primeira fase os modelos HMM estão a segmentar um segmento de áudio de 10 segundos, para identificar possíveis tosses dentro desse segmento; na segunda fase cada candidato é classificado de forma binária: positivo (tosse) ou negativo. A cada ciclo de refinamento os modelos serão actualizados, podendo ou não ser necessários mais ciclos de refinamento.

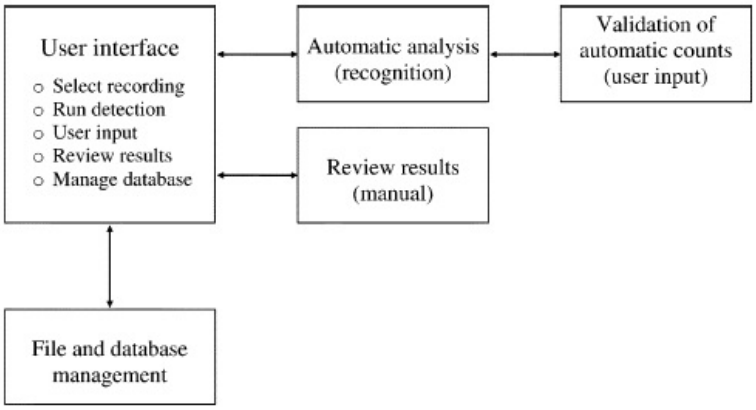

Figura 2 - Vista geral do sistema de análise [4]

A Figura 2 apresenta uma visão geral do sistema de análise, utilizado após a recolha de dados do paciente.

1.1 Objetivos

Este projeto tem como objetivo o desenvolvimento de uma plataforma para a gestão clínica de pacientes com doenças respiratórias que permitirá a integração com o sistema de monitorização e análise sobre a frequência de tosse, LCM. Pretende-se desta forma permitir que esta plataforma seja utilizada de uma forma muito mais difundida, simplificando processos como a submissão das gravações. Com o aparecimento de novos centros que pretendem utilizar este mesmo sistema de monitorização é fulcral a criação de uma plataforma capaz de disponibilizar as suas funcionalidades de forma rápida e eficiente.

1.2 Estrutura do documento

Este documento encontra-se dividido em 6 capítulos com as seguintes temáticas:

Capitulo 1: Introdução e contextualização do trabalho e dos objetivos que visa atingir.

Capitulo 2: Apresentação dos requisitos de sistema a ser tomados em conta para o desenvolvimento do projeto.

Capitulo 3: Descrição das diferentes arquitecturas projectadas para a resolução do projeto. Apresentação da arquitectura escolhida.

Capitulo 4: Soluções tecnológicas – Descrição das tecnologias consideradas para a implementação da arquitectura definida.

Capitulo 5: Implementação do sistema – Apresentação das tecnologias seleccionadas e descrição de todo o processo de desenvolvimento referente ao projeto.

Capitulo 6: Conclusões – Apresentação das conclusões retiradas referentes ao desenvolvimento deste trabalho e possíveis melhorias para trabalhos futuros.

Capítulo 2

Requisitos de sistema

Este capítulo tem como objetivo especificar os requisitos de sistema tendo em conta o desenvolvimento de uma plataforma capaz de integrar as funcionalidades existentes na aplicação LCM num contexto Web, em que não exista necessidade de acesso físico à máquina onde este se encontra implementado. Desta forma foram pensados os requisitos necessários para o desenvolvimento, contemplando a criação e edição de estudos, associados a pacientes, a análise automática desses estudos, e o processo de refinamento dos resultados.

2.1 Modelo de casos de uso

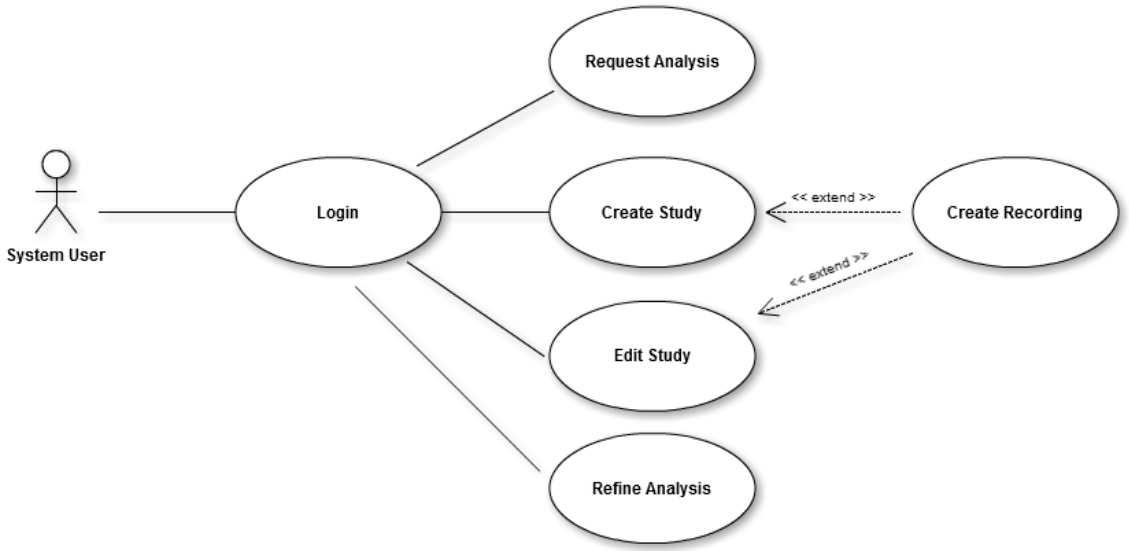

Figura 3 - Modelo de casos de uso

A Figura 3 demonstra quais os casos de uso a ser considerados para o sistema. Estes representam interacções entre o utilizador, identificado por *System User*, e o sistema.

- *Request Analysis* – Proceder à análise do estudo, nomeadamente das gravações obtidas, com recurso ao sistema LCM.
- *Create Study* – Criar um estudo relativo a um paciente, que posteriormente será análise.

- *Edit Study* – Edição da informação referente a um estudo.
- *Refine Analysis* – Realizar refinamento sobre um estudo que já foi analisado.

2.2 Requisitos funcionais

Para que a implementação da plataforma seja considerada bem sucedida é necessário que esta cumpra um conjunto de requisitos. Tendo em conta o modelo de casos de uso apresentado na Figura 3 foram identificados os seguintes requisitos funcionais.

Requisito	Descrição
Adição de novos pacientes	O utilizador deverá ser capaz de adicionar um novo paciente.
Criação de novos estudos	O utilizador deverá ser capaz de criar um novo estudo associado a um paciente.
Adição de novas gravações	O utilizador deverá ser capaz de adicionar gravações áudio a um determinado estudo.
Submissão de ficheiros	O utilizador deverá ser capaz de submeter ficheiros, referentes às gravações.
Armazenamento de ficheiros	A plataforma deverá ser capaz de armazenar ficheiros.
Reprodução de ficheiros áudio	O utilizador deverá ser capaz de reproduzir ficheiros áudio referentes a possíveis eventos de tosse.
Classificação de eventos candidatos a tosse	O utilizador deverá ser capaz classificar possíveis eventos de tosse.

A tabela representa os requisitos funcionais identificados.

2.3 Atores de sistema

O sistema tem como único ator o utilizador, que tem como principal responsabilidade a utilização da aplicação garantindo a qualidade e fidedignidade dos dados introduzidos no sistema.

2.4 Características dos utilizadores

Este sistema destina-se a utilizadores associados à área médica mais concretamente profissionais relacionados com doenças respiratórias.

Capítulo 3

Arquitectura do Sistema

De forma a existir uma concretização bem sucedida da plataforma em questão, é necessário desenvolver uma arquitectura capaz de satisfazer os diversos requisitos da mesma.

Foram então pensadas duas abordagens distintas, nomeadamente um sistema centralizado e um distribuído, sendo que esta nomenclatura foi adoptada principalmente pela forma como cada sistema utiliza a aplicação LCM, que será explicado em seguida.

É importante ter em conta que ambas as abordagens foram pensadas de forma a contemplar uma aplicação Web e um *Web service*. Optou-se por uma aplicação Web devido a não existir necessidade de instalação de nada por parte do utilizador, necessitando apenas de utilizar um Web browser para aceder à aplicação tornando a sua adopção muito mais simplificada, e a utilização de um *Web service* para servir essa mesma aplicação.

3.1 Sistema centralizado

O sistema LCM, conforme existe atualmente, pode ser considerado como um sistema centralizado. No entanto a arquitectura pensada visa realizar uma melhoria nos processos em grande parte do ciclo de vida da aplicação, acrescentando flexibilidade através da descentralização de alguns componentes da aplicação.

O aspecto em destaque face à arquitectura a ser apresentada no sistema distribuído prende-se com a fase em que é realizada a análise sobre as gravações áudio recolhidas do paciente. Neste caso, esta será realizada após a submissão dos ficheiros contendo a gravação.

Este sistema é na sua essência composto por duas partes principais, a aplicação Web onde o utilizador irá efectivamente interagir com o sistema e uma API (*Application Programming Interface*) de serviços web que tratará de toda a parte da lógica de negócio bem como persistência de dados e comunicação com a aplicação LCM.

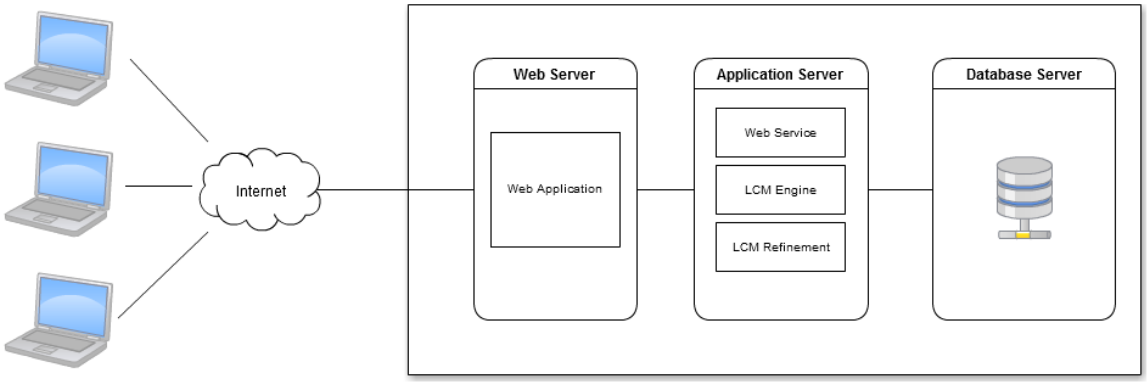

Figura 4 - Sistema Centralizado

Com a utilização desta arquitectura, o sistema passa a poder ser utilizado em qualquer lugar desde que o utilizador possua credenciais válidas para aceder ao mesmo.

O problema identificado relativamente a esta arquitectura é a necessidade da transferência de ficheiros relativamente grandes.

3.2 Sistema distribuído

Com esta arquitectura pretendia-se uma abordagem diferente, tendo sido considerada após terem sido identificados alguns potenciais problemas relativos à implementação da arquitectura anteriormente apresentada. Enquanto o sistema centralizado apresenta uma melhoria relativamente ao que já existe, o sistema distribuído pretendia corrigir alguns dos problemas identificados no sistema centralizado.

Os principais problemas identificados foram a necessidade de transferência de ficheiros de tamanho considerável e todo o processamento/análise dos ficheiros áudio se encontrar centralizado podendo existir falta de recursos para esse processamento. Face a estes problemas foi pensado este sistema, em que existiriam dois tipos de plataformas, uma presente nos centros onde o estudo se estaria a realizar e a outra no centro principal.

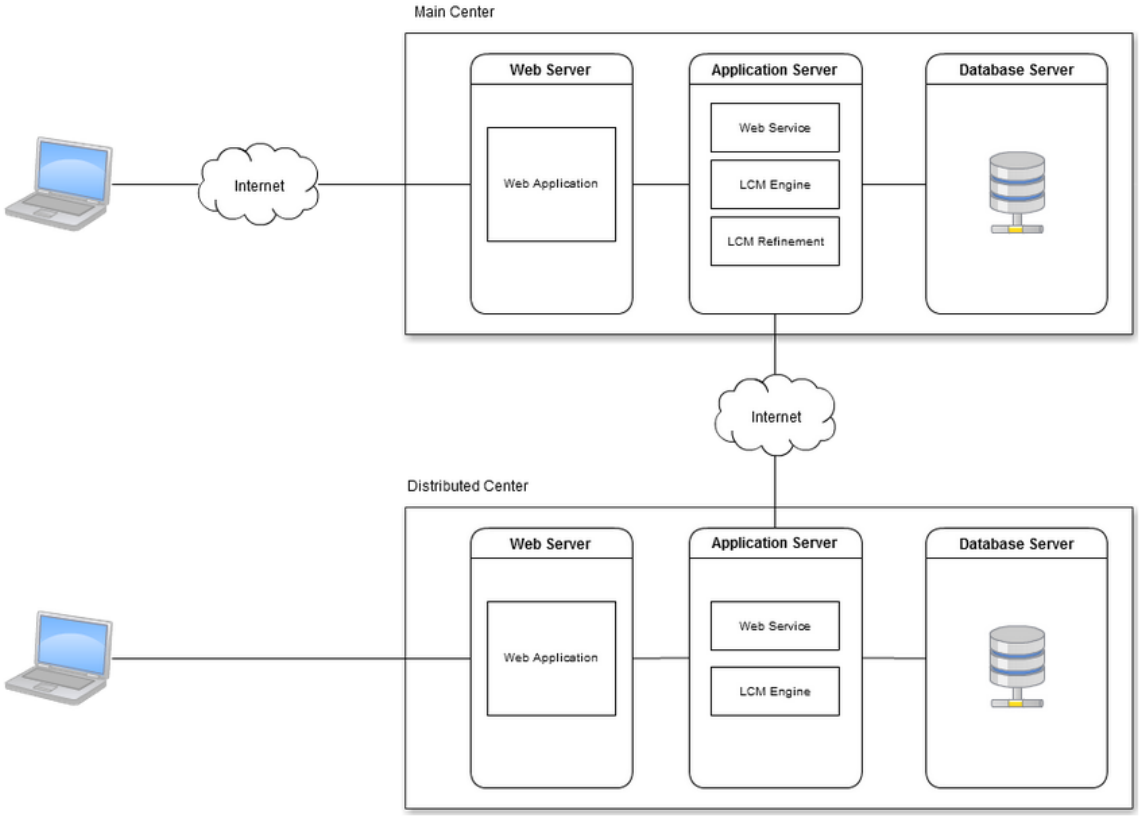

Figura 5 - Sistema distribuído

Neste caso, as plataformas que se encontram nos centros distribuídos não submetem o ficheiro de áudio completo para o centro principal. Ao invés disso seria realizada uma análise na plataforma local, isto utilizando o módulo do LCM, sendo que apenas os ficheiros candidatos a tosse seriam enviados para a plataforma central onde os técnicos procederão à sua classificação.

Desta forma, com este sistema reduziríamos consideravelmente a necessidade de transferência de ficheiros e capacidade de processamento por parte do centro principal.

3.3 Conclusões

Face a tudo apresentado foi optado por implementar a arquitectura sistema centralizado, isto devido à sua maior facilidade de manutenção e implementação. No entanto é necessário ter em conta que devido à sua modularidade facilmente se poderia adaptar colmatando as falhas identificadas, como a submissão de ficheiros de tamanho considerável. Permitirá também um maior controlo sobre a informação uma vez que esta

terá de ser submetida para o centro principal, criando também uma salvaguarda de qualidade.

Capítulo 4

Soluções tecnológicas

Neste capítulo serão discutidas as tecnologias consideradas para implementação do sistema em questão.

4.1. Armazenamento de dados

Para o armazenamento de dados nomeadamente os ficheiros relativos a cada gravação será utilizado o sistema de ficheiros da máquina onde a aplicação estará a correr e para guardar informação foram consideradas diversas tecnologias de base de dados.

4.1.1. Base de dados

Structured Query Language (SQL)

SQL é uma linguagem de programação com um propósito especial que é a gestão de dados que se encontram num sistema de base de dados relacional. Este sistema, como o nome indica assenta num modelo relacional, em que registos de cada tabela se podem relacionar entre si através da definição de uma relação. O modelo baseia-se em dois conceitos, entidade e relação. Uma entidade é caracterizada pelos atributos usados na sua identificação e a relação permite determinar como os registos relacionam entre registos de outras entidades. O objetivo do modelo relacional é fornecer um método declarativo para especificar dados e pesquisas onde os utilizadores especificam directamente que informação a base de dados possui e que informações pretendem receber, onde o sistema de gestão de base de dados é responsável por descrever as estruturas de dados para armazenamento de dados e processos de resposta a pesquisas.

A linguagem SQL é dividida em subconjuntos de acordo com as operações que pretendemos realizar sendo estas compostas por manipulação, definição, controle, transacção e consulta de dados. SQL também composto por procedimentos, ou seja rotinas, subrotinas, métodos e funções.

PostgreSQL [5]

PostgreSQL é um sistema de base de dados objeto-relacional, que possui mais de 15 anos de desenvolvimento activo e uma arquitectura que ganhou uma grande reputação pela integridade de dados e confiabilidade. Segue o modelo ACID (*Atomicity, Consistency, Isolation, Durability*), ou seja possui as propriedades de atomicidade, consistência, isolamento e durabilidade, assegurando fiabilidade no processamento das transações, possui suporte para chaves estrangeiras (*foreign key*s), junções (*joins)*, vistas (*views)*, gatilhos (*triggers)* e *stored procedures*. Possui também diversas interfaces de programação para C/C++, Java, Python, entre outros.

A sua implementação SQL está em conformidade com o padrão ANSI-SQL:2008.

Possui também uma serie de extensões e funcionalidades avançadas, entre elas encontra-se o auto incremento das colunas com utilização de sequencias e a funcionalidade *LIMIT* permitindo obter um resultado parcial.

PostgreSQL pode executar *store procedures* em mais de uma dúzia de linguagens de programação e ainda a sua própria linguagem procedimental. Incluído na sua biblioteca de funções padrão estão centenas de funções embutidas desde matemática básica e operações de *string* a criptografia.

MySQL [6]

MySQL Server é destinado a sistemas de missão crítica com elevada carga de comunicação bem como embutir em softwares distribuídos em massa. É o sistema de gestão de base de dados *open source* mais popular, sendo que é desenvolvido, distribuído e suportado pela corporação Oracle.

MySQL server foi originalmente desenvolvido para lidar com grandes bases de dados de forma mais rápida que as soluções existentes. Apesar de se encontrar em constante desenvolvimento, MySQL Server oferece um variado e útil conjunto de funcionalidades, a sua conectividade, velocidade e segurança tornam o MySQL Server altamente adequado para aceder a bases de dados na Internet.

MySQL Server possui um conjunto de recursos práticos desenvolvidos em cooperação com os seus utilizadores. Desta forma diversas linguagens suportam MySQL Database Server.

SQLite [7]

SQLite é uma biblioteca em tempo de execução que implementa um motor de base de dados SQL transaccional auto-suficiente, sem necessidade de servidor e configuração.

SQLite é um motor de base de dados SQL embutido. Ao contrário da maior parte das bases de dados SQL, SQLite não possui um processo de servidor separado. Este lê e escreve directamente para ficheiros em disco. Uma base de dados SQL completa com varias tabelas, índices, *triggers* e *views*, esta contida num único ficheiro em disco. O formato do ficheiro da base de dados é multi-plataforma, podendo este ser copiado sem problemas. Estas características tornam o SQLite uma escolha popular como formato de ficheiro de aplicação.

SQLite é muito cuidadosamente testado antes de cada lançamento e tem uma reputação por ser muito confiável. As transacções são ACID mesmo se interrompidas por falhas de sistemas ou energia.

SQLite não é directamente comparável com base de dados SQL cliente/servidor uma vez que pretende resolver um problema diferente. Essas bases de dados tem um maior impacto em repositórios partilhados dando maior importância à escalabilidade, concorrência, centralização e controlo enquanto o SQLite fornece um armazenamento local para aplicações individuais e dispositivos dando maior importância à eficiência, fiabilidade, independência e simplicidade.

NoSQL

Uma base de dados NoSQL, originalmente referido por *Non SQL* ou *Non relational* fornece um mecanismo para armazenamento e consulta de dados que é modelado de outra forma que relacional. O surgimento desta abordagem foi motivada pela necessidade por uma maior simplicidade de desenho, escalabilidade horizontal mais simples, sendo este um dos problemas das bases de dados relacionais, e ainda um maior controlo sobre a disponibilidade dos dados, isto devido ao crescimento de soluções *cloud* e a sua necessidade de grande escalabilidade.

Tipicamente soluções NoSQL favorecem disponibilidade, velocidade e tolerância a partição sendo descurado o aspecto da consistência de dados.

Existem diversas abordagens para classificar bases de dados NoSQL, podendo categoriza-las como as seguintes: *Document-oriented, Key-value, Graph, Column* e *Multi-model*. Em seguida serão apresentadas algumas soluções e onde será descrito as especificações para cada abordagem bem como as características da base de dados em questão.

Document-oriented

Document-oriented são por natureza uma subclasse das bases de dados do tipo *key-value*. A diferença reside na maneira como os dados são processados, numa *key-value* os dados são considerados como indiferentes para a base de dados, enquanto num sistema *document-oriented* conta com a estrutura interna do documento para extrair meta dados que o motor da base de dados usa para uma maior optimização.

Este tipo de base de dados contrasta fortemente com a tradicional base de dados relacional. Bases de dados relacionais são fortemente tipadas durante a sua criação e guardam informação semelhante em tabelas separadas que são definidas pelo programador. Todas as instâncias de dados tem o mesmo formato e a alteração deste formato é geralmente difícil. Bases de dados de documentos recebem a informação do tipo dos próprios dados, normalmente guardando toda a informação relacionada em conjunto e permite que cada instância possa ser distinta das restantes. Isto torna-as mais flexíveis relativamente a mudanças e campos opcionais, mapeia mais facilmente em objetos e frequentemente reduz o tamanho da base de dados. Isto torna-las atractivas para aplicações Web modernas que estão sujeitas a constantes mudanças e onde a velocidade de implementação é uma questão importante.

Dados e relações não são armazenados em tabelas como é norma nas bases de dados relacionais mas de facto são uma colecção de documentos independentes.

O facto que estas bases de dados não possuírem uma estrutura definida, adicionar campos a documentos JSON torna-se uma tarefa simples sem necessitar de alterações prévias.

Column-oriented

Column-oriented, os dados são armazenados em células agrupadas em colunas de dados ao invés de linhas de dados. Colunas são logicamente agrupadas em famílias de colunas. Famílias de colunas podem conter um número virtualmente ilimitado de colunas que

podem ser criadas em tempo de execução ou na definição da estrutura. Ler e escrever é realizado usando colunas em vez de linhas.

Em comparação, grande parte dos sistemas de gestão de base de dados relacionais armazenam os dados em linhas, o benefício de guardar dados em colunas, é a sua rápida pesquisa/acesso e agregação de dados. Bases de dados relacionais guardam uma linha como uma entrada continua em disco. Diferentes linhas são guardadas em diferentes locais do disco enquanto bases de dados *column-oriented* guardam todas as células correspondentes a uma coluna numa entrada continua em disco, desta forma tornando a pesquisa/acesso mais rápido.

Key-Value

Key-Value possui um formato sem esquema no entanto possui tudo o que é necessário para satisfazer as necessidades de armazenamento. A chave pode ser criada ou auto-gerada e o seu valor pode ser de diversos tipos como *string* e JSON.

O tipo do valor da chave, basicamente, utiliza uma tabela *hash* na qual existe uma chave única e um apontador para um determinado elemento. Um *bucket* é um grupo lógico de chaves que no entanto não servem para fisicamente agrupar os dados. Podem existir chaves idênticas em diferentes *buckets*.

O desempenho é melhorado em grande parte devido a mecanismos de cache que acompanham os mapeamentos. Para obter um valor é necessário saber a chave e o balde porque a verdadeira chave é um *hash* (*Bucket* + *Key*).

Não há nenhuma complexidade em torno do modelo de base de dados do tipo *key-value*, podendo esta ser implementada sem grande dificuldade. Não é a abordagem ideal se o objetivo for apenas atualizar parte de um valor ou realizar consultas a base de dados.

Graph Base

Base de dados *Graph Base*, não possui o formato rígido de SQL ou a representação em tabelas e colunas. Uma flexível representação gráfica é usada, o que é perfeito para resolver problemas de escalabilidade. Estruturas de grafos são utilizadas com arestas, nós e propriedades fornecendo adjacência livre de índice. Os dados podem ser facilmente transformado de um modelo para outro usando uma base de dados *Graph Base*.

• Estas bases de dados usam arestas e nós para representar e armazenar dados.

• Estes nós são organizados por alguns relacionamentos uns com os outros, que são representados pelas arestas entre os nós.

• Ambos os nós e as relações têm algumas propriedades definidas.

Conclusões

Existem razões suficientes para optar por qualquer dos tipos de bases de dados (SQL e NoSQL). Apesar de estas possuírem o mesmo objetivo, que é armazenar dados, possuem também abordagens distintas, tornando-as alternativas entre si e não uma substituição de SQL por NoSQL como é atribuída frequentemente devido ao seu aparecimento ser relativamente mais recente face ao SQL.

Soluções utilizando NoSQL recebem favoritismo quando possuímos a necessidade de rápido acesso em enormes quantidades de dados devido ao seu alto desempenho de uma forma linear sendo escalável horizontalmente.

Soluções utilizando SQL permitem aplicar uma estrutura rigorosa dificultando e até mesmo impossibilitando o aparecimento de erros de integridade da informação. Estas mesmas opções não estão disponíveis em bases de dados NoSQL, sendo mais flexíveis, permitindo armazenar os dados que se pretendem independentemente das suas relações. No entanto esta capacidade pode levar a problemas de consistência se não for devidamente controlado [8].

Quando é necessário realizar consultas complexas sobre os dados armazenados, bases de dados SQL são uma melhor opção nesses ambientes.

Outro aspecto necessário a ser tomado em conta são as transacções. Enquanto que em ambientes SQL podemos estar a atualizar registos distintos, em tabelas distintas, se estas se encontrarem na mesma transacção garante que ou todas as modificações são efectuadas com sucesso ou então nenhuma é efectuada, garantindo desta forma consistência de informação e controlo de erros. Em ambientes NoSQL, as alterações são atómicas. É possível alterar diversos valores no mesmo documento e estas alterações são garantidas que são todas efectuadas com sucesso ou nenhuma é, no entanto quando se pretende realizar modificações em diversos documentos em simultâneo não é possível garantir esta

consistência de informação, sendo necessário ser processada e garantida programaticamente [9].

Bases de dados NoSQL podem ser mais adequadas para projetos onde os requisitos iniciais sejam difíceis de definir, uma vez que fornecem uma maior flexibilidade, se os dados a ser armazenados se encontram em constante alteração, é necessária uma elevada disponibilidade para lidar com um possível crescimento exponencial ou os dados possuem um crescimento rápido e existe a necessidade de escalar rapidamente. Nestas situações soluções NoSQL tendem a possuir um melhor desempenho devido as suas características. Caso não se preveja alterações estruturais, a integridade dos dados é essencial e o seu crescimento é facilmente controlado então soluções SQL serão mais adequadas [10].

Actualmente, soluções SQL estão a implementar características NoSQL, e vice-versa. A escolha sobre o tipo de base de dados que devemos utilizar cada vez torna-se mais difícil sendo que soluções futuras utilizando características de ambas as abordagens poderão fornecer opções interessantes [11].

Para proceder a escolha da melhor tecnologia a utilizar na implementação da solução a desenvolver é necessário ter muito bem definido o comportamento que é pretendido da base de dados e identificar as características essenciais que se pretende da mesma. Todas as soluções apresentadas tem o propósito de guardar e aceder a informação no entanto foram projectadas com finalidades e comportamentos distintos.

4.1. Web Framework

Uma *framework* em desenvolvimento de software, é uma abstracção composta por um conjunto de conceitos fornecendo uma funcionalidade genérica. Esta visa facilitar o desenvolvimento de aplicações de software, sendo que esta irá determinar o fluxo de controlo da aplicação.

A utilização de uma *framework* possui inúmeras vantagens como uma maior facilidade na detecção de erros, eficiência na resolução de problemas e optimização de recursos. Garante a longevidade das aplicações sendo a sua manutenção orientada também à *framework* e não apenas à aplicação, desta forma sendo mais fácil a integração de outras pessoas no projeto [12].

Uma *framework* para desenvolvimento de aplicações web é uma *framework* de software designado para suportar o desenvolvimento de websites dinâmicos, aplicações Web e serviços Web. A *framework* destina-se a aliviar a sobrecarga associada a actividades comuns realizadas em desenvolvimento Web, promovendo a reutilização de código, seguindo uma abordagem DRY (*Don't Repeat Yourself*).

Sendo que estas podem fornecer uma variadas de funcionalidades como:

- Sistema de template utiliza um processador de template para combinar diferentes templates em uma página Web podendo usar informação para personalizar as páginas.
- Web cache é uma tecnologia que permite o armazenamento temporário de artefactos Web de forma a reduzir a necessidade de realizar alguns pedidos a fontes externas. Algumas *frameworks* possuem mecanismos de *caching* de documentos podendo saltar algumas fases no carregamento da página como o acesso a base de dados.
- Segurança de aplicações Web como o nome indica lida especificamente com segurança em websites, aplicações Web e *Web services*. Algumas *frameworks* possuem *frameworks* de autenticação e autorização que permitem ao servidor Web identificar utilizadores e limitar o acesso a determinados recursos.
- Mapeamento, configuração e acesso à base de dados, muitas *frameworks* possuem uma API unificada para uma base de dados *back-end*, permitindo as aplicações Web trabalhar com uma variedade de base de dados sem efectuar alterações, e também interagir com a base de dados através de uma camada de abstracção.
- Mapeamento URL é o mecanismo pelo qual a *framework* interpreta os URLs, desta forma determinando que recursos deve disponibilizar. Existem diversas formas do mecanismo operar desde expressões regulares a reescrever o URL.
- Asynchronous JavaScript and XML (Ajax) é uma técnica de desenvolvimento web para criar aplicações Web. Tem como objetivo tornar as páginas Web mais responsivas através de diversos pedidos 'não observáveis', isto para que a página inteira não tenha de ser carregada cada vez que existe uma comunicação com o

servidor. Desta forma aumentando a velocidade, interactividade e usabilidade da página.

Apresentadas diversas funcionalidades fornecidas por *web frameworks* passaremos então a apresentar algumas arquitecturas sobre as quais foram desenvolvidas estas *frameworks*.

4.1.1. Arquitectura

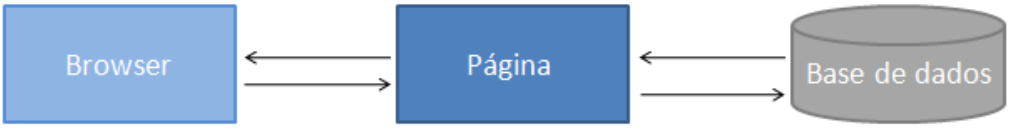

Figura 6 - Arquitectura básica

A Figura 6 demonstra uma arquitectura básica Web.

Model-View-Controller

Model-View-Controller é um conceito para encapsular dados com o seu processamento e isolá-los do processo de manipulação e apresentação. Segue a abordagem mais comum de estratificação, isto representa dividir logicamente o código em funções em diferentes classes. A principal vantagem desta abordagem é a reutilização do código desenvolvido. Seguindo então o modelo MVC, este está dividido em 3 camadas.

Modelo: Gere informação e avisa quando existirem alterações

Vista: é responsável pela representação visual do sistema, pode estar associada a um modelo e quando este atualizar as alterações são aplicadas automaticamente.

Controlador: Recebe interacções do utilizador e actua sobre a vista e o modelo com base na interacção do utilizador.

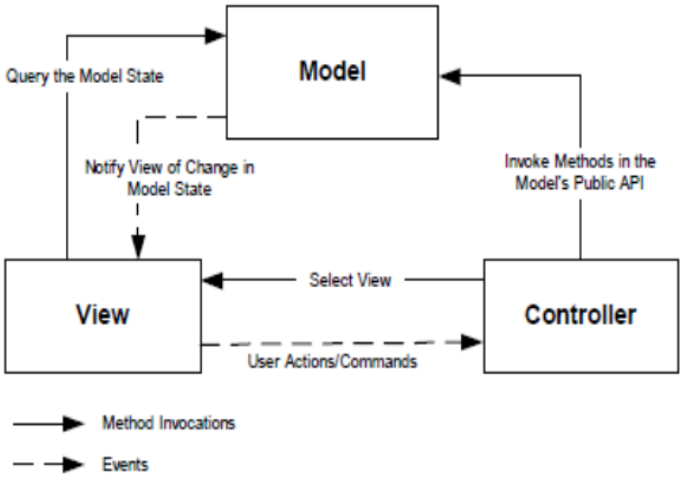

Figura 7 - Arquitectura Model-View-Controller [13]

A utilização de uma arquitectura MVC aumenta a modularidade e reutilização da aplicação existindo uma clara separação entre os diferentes componentes que constituem a aplicação [13].

4.1.2. Lista de frameworks

Baseado em dois rankings foram escolhidas algumas *frameworks* para serem estudadas.

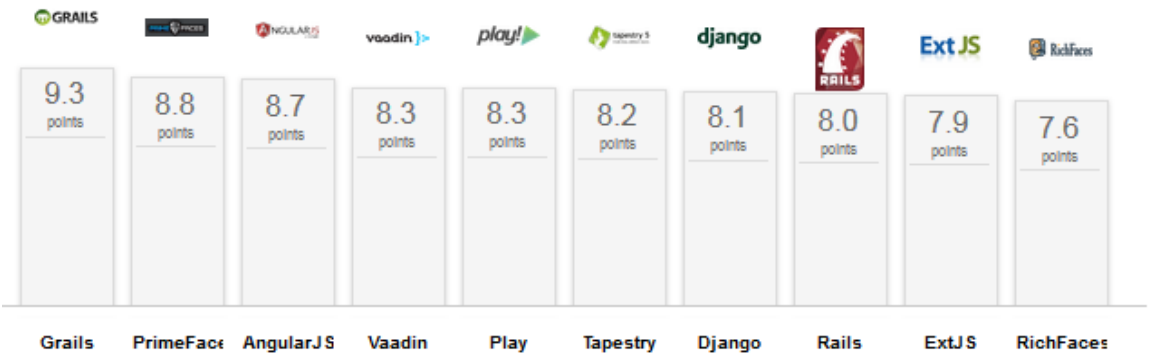

Figura 8 - Ranking de Web frameworks [14]

	Github Score	Stack Overflow Score	Overall Score
ASP.NET		100	100
Ruby on Rails	95	98	96
AngularJS	100	93	96
ASP.NET MVC		93	93
Django	90	92	91

Figura 9 - Ranking de Web frameworks [15]

De entre os rankings apresentados na Figura 8 e Figura 9, foram escolhidas três frameworks, Ruby on Rails, AngularJS e Django.

Ruby on Rails [16]

Ruby on Rails é uma *framework* baseada na linguagem de programação Ruby e que segue o padrão de arquitectura MVC. Segue dois conceitos que visam aumentar a produtividade que são DRY e *Convention over Configuration*.

AngularJS [17]

AngularJS é uma *framework* para desenvolver aplicações Web dinâmicas. Permite a utilização de HTML para criar os templates e estende a sintaxe do HTML para utilizar de forma clara e sucinta os componentes da aplicação. A sua forma de interagir com os dados e a sua injecção de dependência elimina muita da lógica que teria de ser feita para interligar componentes. E uma vez que actua do lado do cliente, é independente de qualquer tecnologia do lado do servidor.

Django [18]

Django é uma *framework* baseada na linguagem de programação Python. O seu principal objetivo é facilitar a criação de websites complexos. O princípio DRY é usado em toda a sua extensão incluindo configurações e modelos de dados.

Conclusões

Com base em todas as *frameworks* estudadas optou-se por criar uma *Single-Page Application* (SPA) com AngularJS, visto ser uma *framework* JavaScript passando grande parte da lógica para o lado do cliente, permitindo assim criar uma aplicação com uma experiência de utilizador superior. Um dos problemas associados à utilização de uma *framework* JavaScript refere-se ao facto destas possuírem uma fraca optimização para motores de busca (*SEO – Search Engine Optimization*). Uma vez que estes motores para realizarem *crawling* (processo que os motores de busca utilizam para percorrer paginas Web com a finalidade de indexa-las correctamente) não se encontram optimizados sobre ficheiros JavaScript. Existem soluções para mitigar este efeito negativo no entanto não passa por uma das necessidades deste projeto [19].

Capítulo 5

Implementação do sistema

Neste capítulo irão ser abrangidos os diferentes processos de implementação deste trabalho. Será explicado como cada componente foi implementado e a razão inerente. Em termos de Web Framework foi optado por utilizar Angular JS, relativamente ao desenvolvimento da API este foi realizado utilizando Java EE. Para a base de dados optou-se por utilizar PostgreSQL 9.4 e como servidor aplicacional Glassfish 4.

Durante a fase de desenvolvimento deste projeto, este teve duas fases distintas. A primeira em que foram desenvolvidos todos os recursos necessários para que a aplicação Web pudesse funcionar correctamente, ou seja, a base de dados e a API. Em seguida foi desenvolvida a aplicação Web, onde o utilizador iria efectivamente interagir com o sistema.

5.1 Armazenamento de dados

A tecnologia adoptada para o desenvolvimento da base de dados foi PostgreSQL devido a algumas funcionalidades já identificadas anteriormente. Destas destacamos a sua reputação no que toca a integridade da informação e o cumprimento do modelo ACID, uma vez estarmos a tratar de dados extremamente sensíveis é crucial satisfazer o aspecto da integridade da informação. O seu suporte por diversas linguagens de programação, o cumprimento do padrão SQL:2008[1] e a possibilidade de adicionar novas funcionalidades personalizadas. Ajudando na escolha da tecnologia foram as extensões que este já possuía como a geração de GUID's e encriptação que serão necessárias para o desenvolvimento do projeto.

5.1.1 Estrutura

Em seguida será apresentada a estrutura implementada em termos de base de dados para dar resposta às necessidades da aplicação.

[1] SQL:2008 representa a sexta revisão da norma ISO e ANSI para a linguagem de consulta SQL

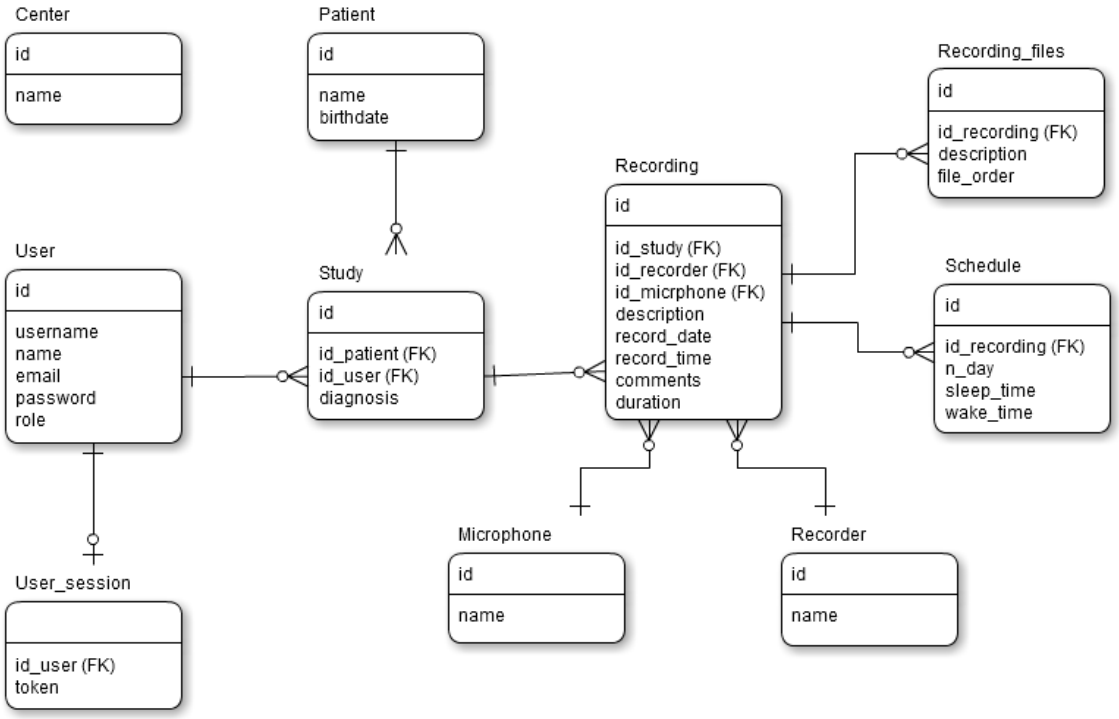

Figura 10 - Modelo ER

Descrição das classes:

User

A classe *User* representa a tabela que contém informação sobre cada utilizador.

Atributo	Descrição
Id	Identificador único para cada utilizador.
Username	Nome de utilizador para acesso a plataforma.
Name	Nome do utilizador.
Email	Email de contacto do utilizador.
Password	Password para aceder a plataforma.
Role	Identifica qual é o tipo de utilizador.

Relação

Um *User* está associado a 0 ou mais *Study*'s.

Um *User* pode ter no máximo um *User_session*.

Patient

A classe *Patient* representa a tabela que contém informação sobre cada paciente.

Atributo	Descrição
Id	Identificador único para cada paciente.
Name	Nome do paciente.
Birthdate	Data de nascimento do paciente.

Relação

Um *Patient* está associado a 0 ou mais *Study*'s.

Study

A classe *Study* representa a tabela que contém informação sobre cada estudo.

Atributo	Descrição
Id	Identificador único para cada estudo.
Id_Patient	Identificador do paciente a que o estudo pertence.
Id_User	Identificador do utilizador que criou o estudo.
Diagnosis	Diagnóstico referente ao estudo.

Relação

Um *Study* é criado por um *User*.

Um *Study* pertence a um *Patient*.

Um *Study* está associado a 0 ou mais *Recording*'s.

Recording

A classe *Recording* representa a tabela que contém informação sobre cada gravação.

Atributo	Descrição
Id	Identificador único para cada gravação.
Id_Study	Identificador do estudo a que a gravação pertence.
Id_Recorder	Identificador do gravador associado a gravação.
Id_Microphone	Identificador do microfone associado a gravação.
Description	Descrição da gravação.
Record_date	Data em que a gravação foi realizada.
Record_time	Hora em que a gravação foi realizada.
Comments	Comentários associados à gravação.
Duration	Duração da gravação.

Relação

Um *Recording* pertence a um *Study*.

Um *Recording* está associado a um *Recorder*.

Um *Recording* está associado a um *Microphone*.

Um *Recording* está associado a 0 ou mais *Schedule*'s.

Um *Recording* está associado a 0 ou mais *Recording_files*'s

Recorder

A classe *Recorder* representa a tabela que contém informação sobre cada gravador.

Atributo	Descrição
Id	Identificador único para cada gravador.
Name	Nome do gravador.

Relação

Um Recorder está associado a 0 ou mais *Recording*'s

Microphone

A classe *Microphone* representa a tabela que contém informação sobre cada microfone.

Atributo	Descrição
Id	Identificador único para cada microfone.
Name	Nome do microfone.

Relação

Um *Microphone* está associado a 0 ou mais *Recording*'s

Schedule

A classe *Schedule* representa a tabela que contém informação sobre o horário de um determinado dia.

Atributo	Descrição
Id	Identificador único para cada horário.
Id_Recording	Identificador da gravação a que o horário pertence.
N_day	Indica a que dia pertence o horário.
Sleep_time	Indica a hora de deitar.
Wake_time	Indica a hora de despertar.

Relação

Um *Schedule* pertence a um *Recording*.

Recording_files

A classe *Recording_files* representa a tabela que contém informação sobre os ficheiros guardados.

Atributo	Descrição

Id	Identificador único para cada ficheiro.
Id_Recording	Identificador da gravação a que o horário pertence.
Description	Indica a descrição do ficheiro.
File_order	Indica a ordem que o ficheiro se encontra relativamente aos restantes.

Relação

Um *Recording_files* pertence a um *Recording*.

Center

A classe *Center* representa a tabela que contém informação sobre cada centro.

Atributo	Descrição
Id	Identificador único para cada centro.
Name	Nome do centro.

5.2 Estrutura da aplicação

5.2.1 Aplicação Web

Relativamente a *Web framework* para desenvolver este projeto foi escolhida a *framework* AngularJS.

Há muitos anos que se utiliza JavaScript nativo juntamente com a biblioteca jQuery para o desenvolvimento de interfaces Web, onde em termos de desenvolvimento e manutenção se tornam em processos complexos e morosos. O uso de uma *framework* JavaScript, como AngularJS, permite focar no desenvolvimento de elementos interactivos para o interface sem grandes preocupações com a estrutura ou manutenção do código.

Esta *framework* garante uma maior escalabilidade, reutilização e manutenção de código JavaScript. Como referido anteriormente, o AngularJS, permite acrescentar atributos HTML com a utilização das suas directivas, estas permitem adicionar comportamento a

elementos DOM ou até mesmo transforma-los, isto de uma forma muito simples. Uma das funcionalidades nucleares do AngularJS é a capacidade de sincronizar informação, quando a informação é alterada na *view* esta também é alterada no *model* e vice-versa.

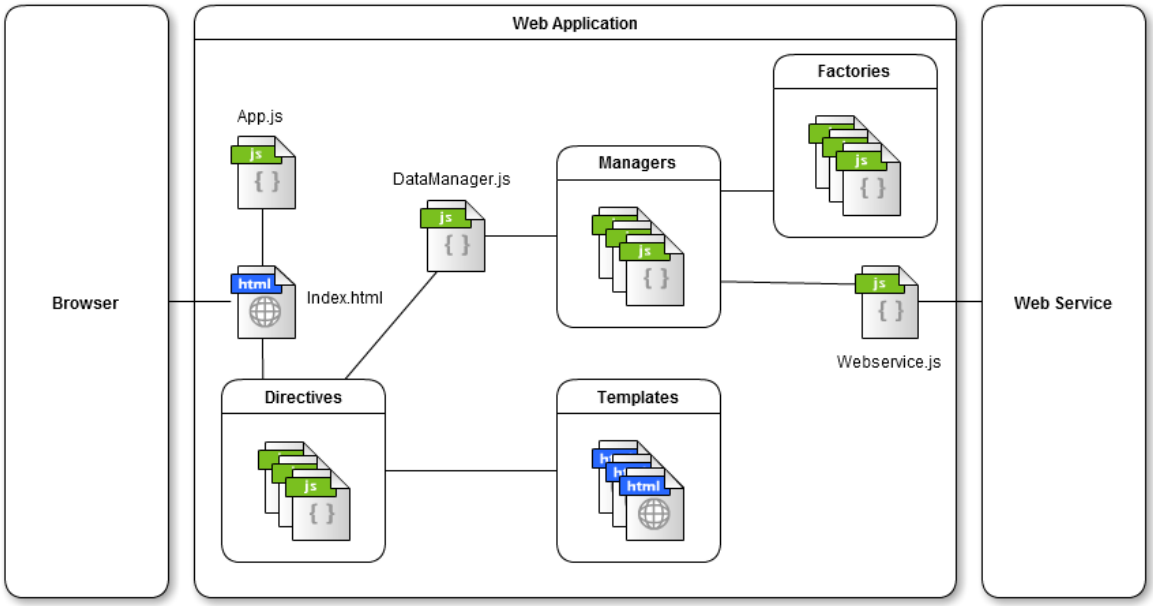

Figura 11 - Arquitectura da Aplicação Web

Desta forma foi possível organizar o desenvolvimento da aplicação por funcionalidades. A Figura 11 mostra a arquitectura implementada na aplicação Web.

A aplicação foi desenvolvida seguindo o princípio de SPA, ou seja toda a interacção é feita na mesma página. No entanto o AngularJS permite gerir *routes*, existindo apenas uma página é através de *routes* que nos é possível navegar pela aplicação. Desta forma podendo apresentar diferentes conteúdos e comportamentos em diferentes *routes* como se de diferentes páginas se tratassem.

A *framework* Angular também permite a criação de directivas, estas representam componentes reutilizáveis que podem ser de diferentes tipos. Podem conter um template HTML associado ou apenas adicionar comportamento a elementos DOM já existentes. Foram criadas diferentes directivas, compostas por um template e lógica de negócio associada, devido a poderem ser reutilizadas torna fácil a sua alteração e manutenção uma vez que seria apenas necessário realizar um mínimo de alterações.

Estas directivas necessitam de apresentar alguns dados, estes são obtidos através do "DataManager", que é um serviço especificamente criado para gerir estes pedidos. Por sua

vez este comunicará com diversos "Managers" associados as diversas entidades como Study e Recording, que efectivamente tratam da disponibilização da informação. Estes "Managers" podem obter os dados de duas formas distintas, através das "Factories" caso os dados já tenham sido previamente carregados e encontrando-se já do lado do cliente ou através de um pedido a um serviço desenvolvido que trata de realizar todos os pedidos ao *Web service*.

5.2.2 Web Service API

Para que este trabalho fosse desenvolvido com sucesso era necessário possuir uma API capaz de dar resposta as necessidades inerentes a toda a aplicação. Sendo que a API disponibiliza a aplicação Web todos as funcionalidades que esta necessita para funcionar. Está encarregue de fornecer toda a informação que necessita para funcionar bem como guardar informação recebida. É também na API que se encontra a integração com o sistema LCM que será explicado em mais detalhe posteriormente neste documento.

A API possui mecanismos de segurança que garantem que apenas utilizadores autenticados tem acesso aos recursos, isto através da verificação do *token*, e que garante que o utilizador que esta a efectuar a comunicação possui acesso ao recurso que esta a tentar aceder, esta validação é efectuada através do seu *role*.

5.2.3 Comunicação

Relativamente a comunicação entre os dois componentes aplicação Web e o *Web service*, e uma vez que não foi utilizada nenhuma *framework* para o desenvolvimento da API procurou-se que esta utiliza-se os métodos http correctos [20] bem como os códigos de resposta adequados [21]. Isto permite que a aplicação Web tenha informação sobre o que ocorreu na comunicação e proceder de acordo com a resposta.

5.2.4 Segurança e autenticação de utilizadores

Para além da necessidade de manter a integridade da informação é também necessário que esta seja apenas acedida por quem possui a devida autorização. Para isso foi então desenvolvido um mecanismo de autenticação garantindo desta forma que o sistema possa verificar a que recursos esse utilizador terá acesso.

Aqui iremos dividir o processo em dois momentos distintos, a autenticação do utilizador e a comunicação quando este já se encontra autenticado.

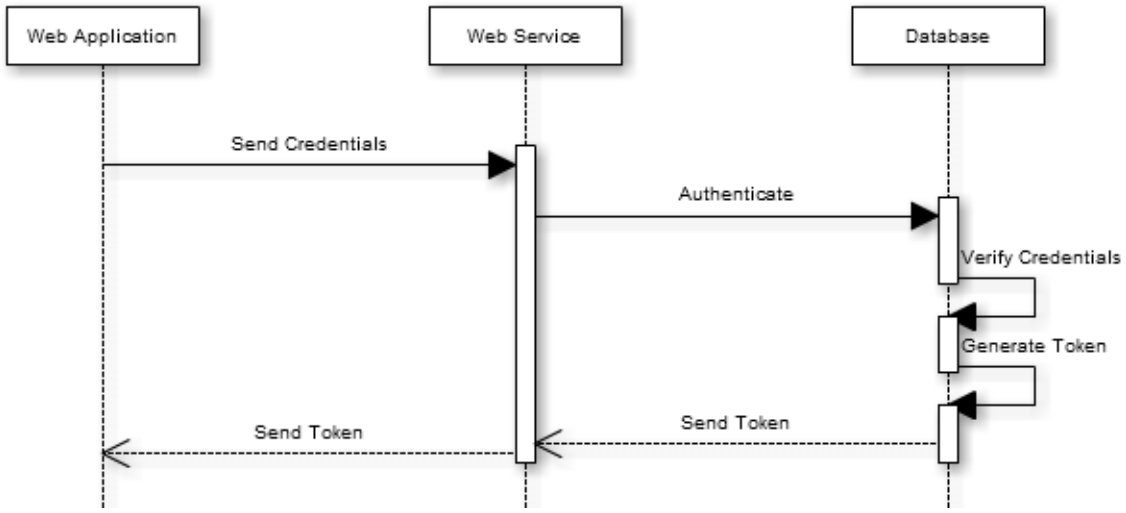

Figura 12 - Fluxo de autenticação

O processo de autenticação, que podemos observar na Figura 12, é constituído por essencialmente quatro momentos:

- Envio de credenciais, introduzidas pelo utilizador.
- Validação das credenciais por parte da base de dados, verificando se existe uma correspondência entre o nome de utilizador e a senha introduzida.
- Geração do *token*, em que irá validar inicialmente se já existe um *token* para aquele utilizador e irá atualizar o mesmo, caso contrário irá criar um novo. A criação do *token* é realizada com auxilio de uma extensão do próprio PostgreSQL chamada de "uuid-ossp" que é uma extensão com a finalidade de gerar identificador únicos universais, sendo para este caso em concreto o método "uuid_generate_v4()".

	uuid_generate_v4 **uuid**
1	08e53fac-7ba9-4f86-8e8e-88d070e89bc7

Figura 13 - Exemplo de token

- Envio do *token*, por fim será enviado o *token* para a aplicação web onde este será guardado para futuras comunicações com o *Web service*.

Quando o utilizador já se encontra autenticado, este passará a enviar o *token* em cada pedido através do campo de cabeçalho do pedido *Authorization*, tendo este sido recebido aquando a autenticação com sucesso.

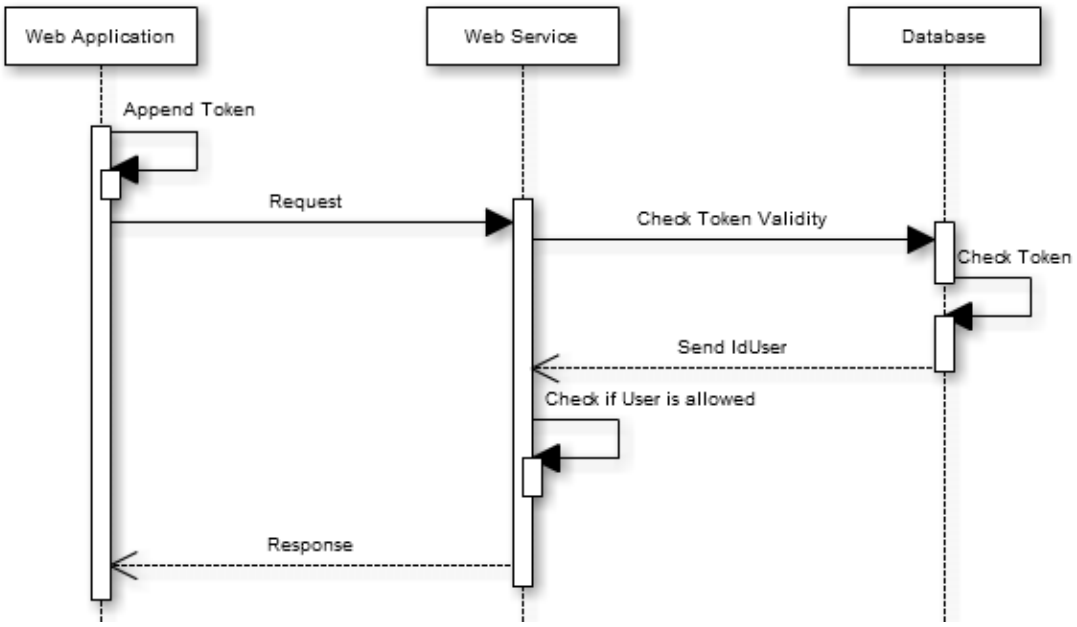

Figura 14 - Fluxo de comunicação

Na Figura 14 podemos observar o fluxo de comunicação quando o utilizador já se encontra autenticado. Antes do pedido ser efectuado é colocado no seu cabeçalho o *token* que deve ser utilizado para a comunicação com o *Web service*, como foi indicado anteriormente. Isto é realizado através da adição de um *Interceptor* na aplicação Web utilizando AngularJS, que concretamente altera todos os pedidos efectuados, adicionando o campo *Authorization* com o *token*, caso este exista [22].

```
.factory('authInterceptor',['SessionService',function(sessionService){
        return {
            'request': function(config){
                    if(sessionService.get('authenticated')){
                        config.headers.Authorization = "Bearer " + sessionService.get('token');
                    } else {
                        config.headers.Authorization = null;
                    }
                    return config;
                }
            };
}])
```

Figura 15 - Criação do interceptor

Na Figura 15 podemos observar de que forma é que o *interceptor* é especificado, verificando-se que apenas os pedidos efectuados pela aplicação serão alterados. Também é

injectado o serviço criado "SessionService", responsável por guardar informação. Para isso este serviço utiliza *SessionStorage*, que assenta basicamente num objeto capaz de guardar informação em pares (chave, valor). Este é criado no browser e apenas é acedido se nos encontrarmos na mesma origem onde este foi criado e tem a particularidade do objeto ser apagado assim que o browser é fechado. Desta forma, previne-se que um simples *refresh* possa obrigar o utilizador a introduzir novamente novas credenciais.

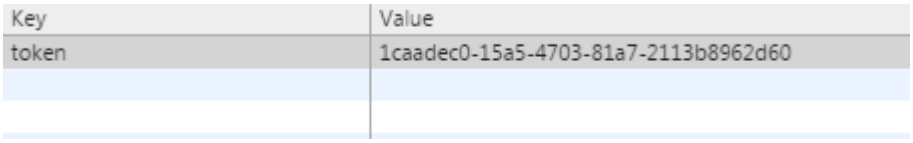

Figura 16 - Exemplo do estado da sessionStorage

A Figura 16 mostra o estado da *SessionStorage* após ter sido efectuado a autenticação do utilizador, guardando desta forma o respectivo *token*.

Para que a aplicação aplique efectivamente o interceptor aos pedidos, é necessário aplica-lo nas configurações da mesma, através da seguinte linha de código:

$httpProvider.interceptors.push('authInterceptor');

Credenciais

Ainda relativamente à questão de segurança, as credenciais são armazenadas encriptadas, de forma a não ser possível obter a senha em questão caso exista uma quebra de segurança.

Explicando o processo de armazenamento das credenciais, assim que é criado um novo utilizador a sua senha é encriptada através da utilização de uma extensão fornecida pela base de dados em PostgreSQL chamada de "pgcrypto". Esta extensão permite encriptar a senha do utilizador através da utilização de um método chamado "crypt". Este método recebe dois parâmetros, a senha a ser encriptada e um *salt*. Um *salt* é utilizado para que caso a mesma senha seja introduzida esta não gere a mesma *hash* encriptada. Este *salt* é criado utilizando outro método chamado "gen_salt" que fornece informação adicional para o método "crypt", indicando o algoritmo a ser utilizado na encriptação e o número de iterações que o algoritmo deve realizar para encriptar a senha. O aumento do número de iterações implica que a encriptação da senha demore mais tempo, o que implica um acréscimo de alguns milissegundos no processo de autenticação. No entanto em caso de quebra de segurança e a *hash* referente a senha do utilizador seja obtido, seriam

necessários milhares de anos (se tiver sido utilizada uma senha alfanumérica) para obter a senha correspondente. O algoritmo utilizado foi o bf (*Blowfish*), uma vez que apresenta diferenças significativas relativamente aos restantes, sendo este o mais seguro [23].

```
INSERT INTO USERS (username,name,email,password, role)
VALUES (
        'Username',                                 -- USERNAME
        'João Costa',                               -- NAME
        'costa.j@ua.pt',                            -- EMAIL
        crypt('password',gen_salt('bf',11)),        -- PASSWORD
        'Admin'                                     -- ROLE
);
```

Figura 17 - Exemplo da inserção de um utilizador

Na Figura 17 podemos observar um exemplo de como seria introduzido um utilizador manualmente.

Sistema de permissões

No sistema desenvolvido existem inúmeros intervenientes, sendo que estes foram classificados em distintos papeis como apresentados anterior. Posto isto e seguindo uma abordagem RBAC foi criado um sistema de permissões baseado nas papéis que cada utilizador. Acabando esta por ser aplicada tanto na aplicação Web, com o objetivo de apenas mostrar as funcionalidades que cada utilizador terá acesso e do lado do *Web service* para garantir que efectivamente apenas utilizadores com as devidas permissões possam executar determinadas acções.

5.2.5 Upload de ficheiros

Um dos requisitos mais importantes do sistema é a capacidade de introduzir ficheiros áudio. É através destes que poderemos realizar análises e apresentar conclusões sobre a condição dos pacientes. Estes são utilizados para alimentar o sistema LCM e produzir tais resultados.

Os ficheiros são carregados para o disco local onde o *Web service* se encontra publicado sendo o seu mapeamento efectuado através dos seus identificadores únicos. Para isto é utilizado o elemento HTML input do tipo *file* (ficheiro) sendo este adicionado a um *FormData* e posteriormente enviado para o *Web service* através de um método *Ajax* do tipo *post* com o encoding *multipart/form-data*.

Foi pensada uma aplicação local com comunicação web para realizar a gestão do carregamento dos ficheiros, à semelhança do serviço disponibilizado pela Dropbox, em que esta também comunicasse com a base de dados podendo fornecer informação sobre o estado do carregamento do ficheiro. Neste cenário seria possível criar mecanismos tolerantes a falhas de conectividade e possibilidade de retomar o carregamento a partir do momento de falha.

5.2.6 Integração do sistema LCM

Um aspecto essencial do trabalho desenvolvido passa pela integração do sistema LCM, pois é este sistema que permite analisar os ficheiros disponibilizados referentes aos pacientes. O sistema LCM foi disponibilizado através de um ficheiro JAR, permitindo assim ser adicionado no nosso *Web service* como uma dependência ao projeto. Para que fosse possível utilizar as funções disponibilizadas pelo sistema LCM foi também necessário adicionar uma dependência adicional, intitulado de *Sphinx*, uma ferramenta *open source* para reconhecimento de fala desenvolvido pela Universidade de Carnegie Mellon [24]. Passando a ser possível retirar partido das funcionalidades disponibilizados pelo sistema LCM num contexto Web.

Análise

O sistema LCM é utilizado como uma aplicação independente pelo que quando seja necessário realizar uma análise, a API instanciará o sistema para este proceder a sua análise. Quando esta análise terminar proceder-se-á à actualização da base de dados de forma a fornecer *feedback* sobre estado da análise. Para que esta análise seja realizada é necessário indicar quais os ficheiros a ser analisados e a localização para onde serão criados os ficheiros temporários de análise e onde eventualmente serão guardados ficheiros relativos a potenciais eventos candidatos a tosse para futuramente serem analisados através do componente de refinamento da análise.

Refinamento

Neste processo serão disponibilizados diversos excertos áudio ao utilizador para que este os possa classificar positivamente ou negativamente se representam eventos de tosse. Para o refinamento o processo diferencia ligeiramente do anterior uma vez que este será um

processo continuo com diversos inputs por parte do utilizador a medida que este classificação os eventos candidatos, será necessário manter o contexto relativamente ao estado do refinamento. Para isso ao iniciar o processo de refinamento é criado um objeto na API que ficará guardado em memória e actualizado periodicamente até que o processo de refinamento seja concluído. Esta classificação irá fornecer informação ao sistema LCM que atualizará os seus modelos e verificará se já se encontra em condições de apresentar conclusões relativamente aos ficheiros análises anteriormente.

Capítulo 6

Conclusões e trabalho futuro

6.1 Conclusões

Através da arquitectura implementada é possível verificar que se trata de um sistema extremamente modular, com as responsabilidades de cada funcionalidade muito bem definidas. Em caso de alteração ao sistema futuro, estas foram devidamente pensadas e ponderadas na elaboração da arquitectura. A aplicação Web e o serviço Web não estão mutuamente dependentes, sendo que cada um deles poderia ser substituído ou alterado sem necessidade de realizar alterações na outra componente. O mesmo se verifica com a base de dados, no entanto esta teria a particularidade de que o serviço Web teria de ser alterado devido à utilização de extensões específicas dos PostgreSQL.

Também existe a possibilidade de acrescentar funcionalidades à aplicação para suportar a análise de outras doenças com base na actual estrutura, com a devida integração de sistemas capazes de realizar essas análises.

A aplicação também suportará alteração ao nível dos processos institucionais, onde poderão existir estudos compartilhados.

6.2 Trabalho futuro

À medida que o desenvolvimento foi avançando foram sendo identificados alguns aspectos passíveis de alteração, quer a nível tecnológico como estrutural.

Ao nível do serviço Web, apesar de este satisfazer o que se pretende, foi desenvolvido a medida que era necessário com o crescimento da estrutura desenvolvida com recurso a *framework* AngularJS. Uma melhoria seria utilizar uma *framework* para o desenvolvimento do serviço Web, aumentando o seu desempenho, segurança e manutenção da solução. Outro aspecto a ser melhorado em relação ao serviço Web seria a implementação de um *interceptor* sobre os pedidos HTTP, desta forma através do *interceptor* este validaria o acesso e permissões aos recursos que se esta a requisitar, criando um controlo mais centralizado e de fácil alteração.

Tal como indicado anteriormente é necessário proceder a alteração do método de *upload* de ficheiros, criando um maior controlo sobre esse processo sendo este um dos requisitos críticos de toda aplicação.

De momento todos os utilizadores podem aceder a todos os estudos, facilmente se poderia realizar a alteração para restringir o acesso aos estudos. Isto seria também relevante para um objetivo futuro, que passa pela criação de estudos que possam ser partilhados e compostos por diversas análises em pacientes distintos.

Bibliografia

[1] European Respiratory Society, “The economic burden of lung disease,” in *European lung white book*, 2003, pp. 16–27.

[2] D. World, “Disabled World: Disability News & Information.” [Online]. Available: http://www.disabled-world.com/health/respiratory/.

[3] S. Matos, S. S. Birring, I. D. Pavord, and D. H. Evans, “Detection of cough signals in continuous audio recordings using hidden Markov models.,” *IEEE Trans Biomed Eng*, vol. 53, pp. 1078–83, 2006.

[4] S. Matos, S. S. Birring, I. D. Pavord, and D. H. Evans, “An automated system for 24-h monitoring of cough frequency: the leicester cough monitor.,” *IEEE Trans. Biomed. Eng.*, vol. 54, pp. 1472–9, 2007.

[5] “PostgreSQL.” [Online]. Available: http://www.postgresql.org/.

[6] “MySQL.” [Online]. Available: https://www.mysql.com/.

[7] “SQLite.” [Online]. Available: https://www.sqlite.org/.

[8] V. Sharma and M. Dave, “SQL and NoSQL Databases,” *Int. J. Adv. Res. Comput. Sci. Softw. Eng.*, vol. 2, pp. 20–27, 2012.

[9] L. P. Issac, “SQL vs NoSQL Database Differences Explained with few Example DB,” 2014. [Online]. Available: http://www.thegeekstuff.com/2014/01/sql-vs-nosql-db/.

[10] E. McNulty, “SQL vs. NoSQL- What You Need to Know,” 2014. [Online]. Available: http://dataconomy.com/sql-vs-nosql-need-know/.

[11] C. Buckler, “SQL vs NoSQL: The Differences,” 2015. [Online]. Available: http://www.sitepoint.com/sql-vs-nosql-differences/.

[12] SensioLabs, “Why should I use a framework?” [Online]. Available: http://symfony.com/why-use-a-framework.

[13] S. Uyun and M. Rifqi, “IMPLEMENTATION OF MODEL VIEW CONTROLLER (MVC) ARCHITECTURE ON BUILDING WEB-BASED INFORMATION SYSTEM,” in *Islam Zeitschrift Für Geschichte Und Kultur Des Islamischen Orients*, 2010, vol. 2010, pp. 47–50.

[14] “Top 10 frameworks.” [Online]. Available: http://devrates.com/stats/.

[15] “Web frameworks ranking.” [Online]. Available: http://hotframeworks.com/.

[16] “Ruby on Rails.” [Online]. Available: http://www.rubyonrails.org/.

[17] “AngularJS.” [Online]. Available: https://angularjs.org/.

[18] “Django.” [Online]. Available: https://www.djangoproject.com/.

[19] C. McKeachie, “Should You Use a JavaScript MVC Framework to Build Your Web Application?,” 2014. [Online]. Available: https://www.codementor.io/javascript/tutorial/should-you-build-your-web-application-with-javascript-mvc-frameworks.

[20] W3C, “HTTP/1.1: Method Definitions.” [Online]. Available: http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html.

[21] W3C, “HTTP/1.1: Status Code Definitions.” [Online]. Available: http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html.

[22] M. Zupan, “Angular Default Request Headers and Interceptors,” 2014. [Online]. Available: http://www.devblogrbmz.com/angular-default-request-headers-and-interceptors/.

[23] J. Atwood, “You’re Probably Storing Passwords Incorrectly,” 2007. [Online]. Available: http://blog.codinghorror.com/youre-probably-storing-passwords-incorrectly/.

[24] P. Lamere, P. Kwok, E. Gouvea, B. Raj, R. Singh, W. Walker, M. Warmuth, and P. Wolf, “The CMU SPHINX-4 speech recognition system,” in *IEEE Intl. Conf. on Acoustics, Speech and Signal Processing (ICASSP 2003), Hong Kong*, 2003, vol. 1, pp. 2–5.